

# Improving the Accuracy of Cache-Aware Response Time Analysis Using Preemption Partitioning

Filip Marković 

Mälardalen University, Västerås, Sweden  
filip.markovic@mdh.se

Jan Carlson

Mälardalen University, Västerås, Sweden  
jan.carlson@mdh.se

Sebastian Altmeyer

University of Augsburg, Germany  
altmeyer@informatik.uni-augsburg.de

Radu Dobrin

Mälardalen University, Västerås, Sweden  
radu.dobrin@mdh.se

---

## Abstract

Schedulability analyses for preemptive real-time systems need to take into account cache-related preemption delays (CRPD) caused by preemptions between the tasks. The estimation of the CRPD values must be sound, i.e. it must not be lower than the worst-case CRPD that may occur at runtime, but also should minimise the pessimism of estimation. The existing methods over-approximate the computed CRPD upper bounds by accounting for multiple preemption combinations which cannot occur simultaneously during runtime. This over-approximation may further lead to the over-approximation of the worst-case response times of the tasks, and therefore a false-negative estimation of the system's schedulability. In this paper, we propose a more precise cache-aware response time analysis for sporadic real-time systems under fully-preemptive fixed priority scheduling. The evaluation shows a significant improvement over the existing state of the art approaches.

**2012 ACM Subject Classification** Computer systems organization → Real-time system specification; Software and its engineering → Real-time schedulability

**Keywords and phrases** Real-time systems, Fixed-Priority Preemptive Scheduling, Preemption delay

**Digital Object Identifier** 10.4230/LIPIcs.ECRTS.2020.5

**Acknowledgements** We want to thank our colleagues Sebastian Hahn, Jan Reineke, and Darshit Shah, who provided us with the evaluation data derived from the code-level analysis of benchmark programs. Also, we are very grateful to Davor Ćirkinagić who borrowed his powerful computing system for performing schedulability evaluation.

## 1 Introduction

Fully-preemptive scheduling is used in many real-time embedded systems in order to e.g., overcome the limitations of non-preemptive scheduling which can introduce significant blocking on high priority tasks from lower priority ones. Fully-preemptive scheduling allows for an interruption (preemption) of the task's execution whenever a task with a higher priority is released. However, as shown by Pellizzoni et al. [21], a preemption can introduce a significant preemption related delay, even up to 33% of the task's worst-case execution time.

In embedded systems employing a cache-based architecture, one of the major causes of preemption delay is cache-related preemption delay (CRPD), as shown by Bastoni et al. [7]. CRPD represents the longest time needed by a resuming task to reload the memory cache blocks which it had loaded prior to the preemption. Since CRPD may significantly increase



© Filip Marković, Jan Carlson, Sebastian Altmeyer, and Radu Dobrin;  
licensed under Creative Commons License CC-BY

32nd Euromicro Conference on Real-Time Systems (ECRTS 2020).

Editor: Marcus Völp; Article No. 5; pp. 5:1–5:23



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the worst-case execution time of a task, its tight estimation is very important and therefore a new field of timing analysis, called cache-related preemption delay analysis, has emerged in the research of real-time systems.

In the context of CRPD analysis for fixed-priority fully-preemptive scheduling, many different approaches have been proposed in the last few decades, of which we describe a selection of the most recent ones. Tomiyama et al. [31], and Busquets-Mataix et al. [11] proposed analyses which are based on the over-approximation that a single preemption causes the CRPD equal to the time needed for reloading all the evicting cache blocks from a preempting task. These analyses neglected the fact that not every eviction results in a cache block reload. Contrary to this, Lee et al. [20] proposed the analysis that bounds the CRPD by accounting for the cache blocks which may be reused at some later point in a task, called the useful cache blocks. However, their analysis did not account for the fact that although useful, some cache block cannot be evicted, and thus cannot result in a cache block reload. These two opposite approaches defined the two main branches in CRPD analysis: ECB-based CRPD and UCB-based CRPD.

Later, Tan and Mooney [30] proposed the UCB-union approach, which accounted for the limitations of the above-described approaches. In this approach, CRPD is computed using the information about all possibly affected useful cache blocks along with the evicting cache blocks from the tasks which may evict them. Opposite to that, Altmeyer et al. [3] proposed the ECB-union approach, where the all possibly evicting cache blocks are analysed along with the useful cache blocks from the tasks that may be preempted.

The latest and in overall the most precise CRPD approaches are proposed by Altmeyer et al. [4], called ECB-union multiset and UCB-union multiset approaches. Those approaches are improvements over the UCB-union and ECB-union because they account for a more precise estimation of the nested preemptions. The multiset approach was also used by Staschulat et al. [28] for the periodic task systems where they accounted that each additionally accounted preemption of a single preempting task may result in a smaller CRPD value compared to the previous preemptions. In the context of periodic systems, Ramaprasad and Mueller [23, 22] investigated the possibility of tightening the CRPD bounds using preemption patterns. However, in this paper, we consider a sporadic task model which constrains such analysis.

Furthermore, in recent years, several cache-aware analysis were proposed in the contexts of: cache partitioning by Altmeyer et al. [26, 5], cache-persistence by Rashid et al. [25, 24] and Stock et al. [29], write-back cache by Davis et al. [12] and Blaß et al. [9].

In all of the above-mentioned CRPD analyses, the resulting upper bounds are overly pessimistic mainly because they account for CRPD obtained from preemption combinations which cannot occur simultaneously during runtime.

In this paper, we propose a cache-aware response-time analysis that accounts for the above-mentioned source of pessimism, and a few more, in the context of fixed priority fully-preemptive scheduling (FPPS). The evaluation shows a significant improvement over the existing state of the art approaches.

In the remainder of the paper, we first define the system model in Section 2. In Section 3, we overview the existing SOTA UCB-union and ECB-union based methods, including the multiset variants. In Section 4, we discuss the pessimism in the current state of the art cache-aware analyses. The proposed analysis is defined in Section 5, and the evaluation results are shown in Section 6. The paper is concluded in Section 7.

## 2 Task Model, Terminology and Notation

In this paper, we consider a sporadic task model, with preassigned fixed task priorities, under fully-preemptive scheduling. A taskset  $\Gamma$  consists of  $n$  tasks sorted in a decreasing priority order, where each task  $\tau_i$  generates an infinite number of jobs, characterised with the following task parameters  $\langle P_i, C_i, T_i, D_i \rangle$ . Task priority is denoted with  $P_i$  and we assume disjunct priorities among the tasks. The worst-case execution time without accounted preemption delays is denoted with  $C_i$ .  $T_i$  denotes the minimum inter-arrival time between the two consecutive jobs of  $\tau_i$ , and the relative deadline is denoted with  $D_i$ .

We also consider single-core systems with single-level direct-mapped caches, extending the task model by accounting for detailed knowledge about the cache usage. In addition, we describe a possible adjustment in subsection 5.8, thus also considering LRU set-associative caches. For each task  $\tau_i$ , the information about the accessed cache blocks within the task's execution is assumed as derived, and based on that we define the following cache block sets:  $ECB_i$  – a set of *evicting cache blocks* of  $\tau_i$ , such that cache-set  $s$  is in  $ECB_i$  if and only if a memory block from  $s$  may be accessed during the execution of  $\tau_i$ .

$UCB_i$  – a set of all *useful cache blocks* throughout the execution of  $\tau_i$ . As proposed by Lee et al. [20], and superseded by Altmeyer et al. [2], a cache-set  $s$  is in  $UCB_i$ , if and only if  $\tau_i$  accesses a memory block  $m$  in  $s$  such that: a)  $m$  must be cached at some program point  $\mathcal{P}$  in the execution of  $\tau_i$ , and b)  $m$  may be reused on at least one control flow path starting from  $\mathcal{P}$  without the eviction of  $m$  on this path.

In the remainder of the paper, we use the following notations for different sets of tasks:

- $hp(i)$  A set of tasks with priorities higher than  $\tau_i$
- $hpe(i)$  A set of tasks with priorities higher than  $\tau_i$ , including  $\tau_i$ , i.e.  $hpe(i) = hp(i) \cup \{\tau_i\}$
- $lp(i)$  A set of tasks with priorities lower than  $\tau_i$
- $aff(i, h)$  A set of tasks with priorities higher than or equal to  $\tau_i$  and lower than  $\tau_h$ , i.e.  $aff(i, h) = hpe(i) \cup lp(h)$

## 3 Background

Cache-related preemption delay is computed as the upper bound on the number of cache block reloads that can be caused due to preemptions and potential evictions of memory contents that are used by the preempted tasks. In this paper, CRPD is denoted with  $\gamma$  and is computed as the multiplication of the upper bound on cache block reloads with the constant  $BRT$ , which is the longest time needed for a single memory block to be reloaded into cache memory, i.e. block reload time. The general formula is  $\gamma = \#reloads \times BRT$ .

In this section, we briefly describe the most relevant CRPD-aware analyses for understanding the contributions of this paper. We describe *UCB-union* approach [30], *ECB-union* approach [3], and their multiset variants [4].

*UCB-union* and *ECB-union* approaches are computed as the least-fixed points of the following equation for the worst-case response time:

$$R_i^{(l+1)} = C_i + \sum_{\tau_h \in hp(i)} \lceil R_i^{(l)} / T_h \rceil \times (C_h + \gamma_{i,h}) \quad (1)$$

In Equation 1,  $\gamma_{i,h}$  represents the CRPD due to a single job of a higher priority task  $\tau_h$  executing within the worst-case response time of task  $\tau_i$ . This term is computed differently for each of the CRPD approaches.

*UCB-Union approach* computes  $\gamma_{i,h}^{ucbu}$  with the following equation, accounting that a job of  $\tau_h$  causes a reload of each cache block which it may access and which is useful during the execution of at least one of the tasks from the range  $[\tau_{h+1}, \tau_{h+2}, \dots, \tau_i]$ .

$$\gamma_{i,h}^{ucbu} = \left| \left( \bigcup_{\tau_k \in \text{aff}(i,h)} UCB_k \right) \cap ECB_h \right| \times BRT \quad (2)$$

*ECB-Union approach* computes  $\gamma_{i,h}^{ecbu}$  with the following equation, accounting that a job of  $\tau_h$  is preempted by all of the tasks with higher priority than  $\tau_h$ , after the job directly preempted one of the tasks from the range  $[\tau_{h+1}, \tau_{h+2}, \dots, \tau_i]$ . In this case, the preemption resulting in the highest number of evicted useful cache blocks is considered.

$$\gamma_{i,h}^{ecbu} = \max_{\tau_k \in \text{aff}(i,h)} \left\{ \left| \left( \bigcup_{\tau_{h'} \in \text{hpe}(h)} ECB_{h'} \right) \cap UCB_k \right| \right\} \times BRT \quad (3)$$

Improving the above two approaches, Altmeyer et al. [4] introduced *UCB-Union multiset* and *ECB-Union multiset* which are computed as the least-fixed points of the following equation:

$$R_i^{(l+1)} = C_i + \sum_{\tau_h \in \text{hp}(i)} (\lceil R_i^{(l)} / T_h \rceil \times C_h + \gamma_{i,h}) \quad (4)$$

where  $\gamma_{i,h}$  represents the CRPD due to each job of a higher priority task  $\tau_h$  executing within the the worst-case response time of task  $\tau_i$ .

*ECB-Multiset approach* computes  $\gamma_{i,h}^{ecbum}$  accounting that  $\tau_h$  can preempt each task  $\tau_k \mid h < k \leq i$  the maximum number of times a single job of  $\tau_k$  can be preempted by jobs of  $\tau_h$ , for each job of  $\tau_k$  that can be released within  $R_i$ , i.e.  $\lceil R_k / T_h \rceil \times \lceil R_i / T_k \rceil$  times. This is accounted by the multiset  $M_{i,h}$ , which consists of the maximum CRPDs from jobs of  $\tau_h$  on each preemptable job which can be released within  $R_i$  ( $\uplus$  represents multiset union):

$$M_{i,h} = \biguplus_{\tau_k \in \text{aff}(i,h)} \left( \biguplus_{\lceil R_k / T_h \rceil \times \lceil R_i / T_k \rceil} |UCB_k \cap \left( \bigcup_{\tau_{h'} \in \text{hpe}(h)} ECB_{h'} \right)| \right) \quad (5)$$

Based on the above multiset,  $\gamma_{i,h}^{ecbum}$  is computed as the sum of the maximum  $\lceil R_i / T_h \rceil$  values from  $M_{i,h}$ , accounting that only  $\lceil R_i / T_h \rceil$  jobs of  $\tau_h$  can directly preempt and cause CRPD on the preemptable jobs accounted in  $M_{i,h}$ .

*UCB-Multiset approach* computes  $\gamma_{i,h}^{ucbum}$  by first computing the multiset  $M_{i,h}^{ucb}$  which consists of all possibly useful cache blocks from jobs which can be released within  $R_i$ , and have priority higher than or equal to  $\tau_i$ , and lower than  $\tau_h$ .

$$M_{i,h}^{ucb} = \biguplus_{\tau_k \in \text{aff}(i,h)} \left( \biguplus_{\lceil R_k / T_h \rceil \times \lceil R_i / T_k \rceil} UCB_k \right) \quad (6)$$

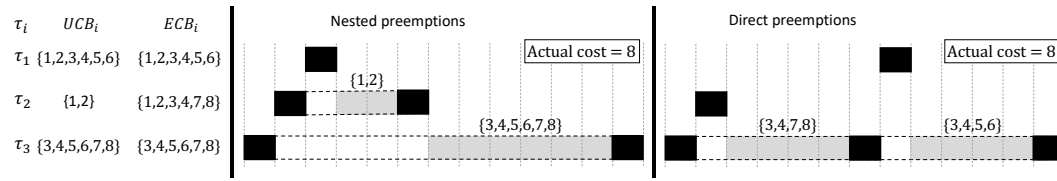
Next, this approach computes the multiset  $M_{i,h}^{ecb}$  which consists of all possibly evicting cache blocks within jobs of  $\tau_h$  that can be released within  $R_i$ . The following equation includes an instance of evicting cache block from  $\tau_h$  for each job of  $\tau_h$  that can be released within  $R_i$ :

$$M_{i,h}^{ecb} = \biguplus_{\lceil R_i / T_h \rceil} (ECB_h) \quad (7)$$

The upper bound on CRPD from jobs of  $\tau_h$  preempting all jobs from  $\tau_{h+1}$  to  $\tau_i$  is equal to the size of intersection of those multisets, with accounted block reload time:

$$\gamma_{i,h}^{ucbum} = |M_{i,h}^{ucb} \cap M_{i,h}^{ecb}| \times BRT \quad (8)$$

The Combined-Multiset approach first computes the worst-case response time  $R_i^{ecbum}$  using Equation 4 and  $\gamma_{i,h}^{ecbum}$ , and similarly does with UCB-Union multiset, using Equation 4 and  $\gamma_{i,h}^{ucbum}$  thus deriving  $R_i^{ucbum}$ . Then, the final result is computed as  $\min(R_i^{ecbum}, R_i^{ucbum})$ .



■ **Figure 1** Example of the pessimistic CRPD estimation in both, UCB- and ECB-union based approaches. Notice that the worst-case execution time (black rectangles) is in reality significantly larger than CRPD, but the focus of the figure is rather on preemptions and CRPD depiction.

#### 4 Pessimism in CRPD analyses based on UCB- and ECB-union approaches

In this section, we present the identified problems considering CRPD over-approximation when using UCB- and ECB-based approaches, including the multiset variants.

► **Problem 1.** *Combined approach over-approximates the CRPD bounds because all preemptions that may occur within a response time  $R_i$  are treated the same, with at most one method at a time. However, within all preemptions that may occur within  $R_i$ , different preemption sub-groups may be analysed with different analyses, thus the CRPD may be further reduced by computing the bounds for different preemption sub-groups individually instead of computing it with one method at a time for a single group of all preemptions.*

► **Problem 2.** *Combined approach accounts for CRPD from many different preemption combinations, which cannot occur together. This is presented with the following example. In Figure 1, we present three tasks  $\tau_1, \tau_2$ , and  $\tau_3$  with their respective sets of evicting and useful cache blocks. In the example, it is assumed that tasks  $\tau_1$  and  $\tau_2$  can be released at most once during the execution of  $\tau_3$  and that block reload time is equal to 1. Based on this, only two preemption combinations which result in the worst-case CRPD bound are possible: 1) A job of  $\tau_2$  directly preempts a job of  $\tau_3$ , and a job of  $\tau_1$  directly preempts a job of  $\tau_2$  (nested preemptions), 2) A job of  $\tau_2$  directly preempts a job of  $\tau_3$ , and a job of  $\tau_1$  directly preempts a job of  $\tau_3$  (direct preemptions).*

For each task, black rectangles in the figures represent the worst-case execution time, grey rectangles represent CRPD, whereas the sets of integer values above the grey rectangles represent the cache sets whose reloads must be accounted.

Considering the given cache block sets, the actual worst-case CRPD, based on the separately analysed preemption combinations, is:

- 8 (nested preemption): This is the case because  $\tau_2$  evicts cache blocks 3, 4, 7, and 8 which are then reloaded during the post-preemption execution of  $\tau_3$ . After that,  $\tau_1$  evicts blocks 1 and 2 when preempting  $\tau_2$ , which are reloaded during the post-preemption execution of  $\tau_2$ , and also  $\tau_1$  evicts cache blocks 5 and 6 which are reloaded during the post-preemption execution of  $\tau_3$ . Notice that although  $\tau_1$  also potentially evicts blocks 3, and 4 from  $\tau_3$ , they are accounted as reloads only once within  $\tau_3$ , because  $\tau_3$  is interrupted once and thus only one reload of each useful cache block within remaining execution of  $\tau_3$  is possible.
- 8 (direct preemptions): This is the case because  $\tau_2$  evicts cache blocks 3, 4, 7, and 8 from  $\tau_3$ , and  $\tau_1$  evicts cache blocks 3, 4, 5, and 6 from  $\tau_3$ .

Since any other preemption combination can be derived only by removing one of the preemptions accounted in the two above, the worst-case CRPD is equal to 8. However,

UCB-union based approaches (including the multiset variant) compute the following CRPD:

$$\begin{aligned}\gamma_{i,h}^{ucbu} &= \left| \left( \bigcup_{\tau_k \in \text{aff}(i,h)} UCB_k \right) \cap ECB_h \right| \\ \gamma_{3,1}^{ucbu} &= \left| (UCB_3 \cup UCB_2) \cap ECB_1 \right| = 6, \quad \gamma_{3,2}^{ucbu} = \left| UCB_3 \cap ECB_2 \right| = 4 \\ \gamma_{3,1}^{ucbu} + \gamma_{3,2}^{ucbu} &= 4 + 6 = 10, \quad \text{accounted reloads for blocks: } 1, 2, 3, 3, 4, 4, 5, 6, 7, 8\end{aligned}$$

UCB-union based approaches compute CRPD upper bound of 10 reloads, thus approximating two block reloads over the safe upper bound (8 reloads) illustrated in Figure 1. Compared to the leftmost case from Figure 1, the accounted infeasible reloads are for blocks 3 and 4. Compared to the rightmost case, the accounted infeasible reloads are for blocks 1 and 2.

ECB-union based approaches (including the multiset variant) compute the CRPD upper-bound as follows:

$$\begin{aligned}\gamma_{i,h}^{ecbu} &= \max_{\tau_k \in \text{aff}(i,h)} \left\{ \left| \left( \bigcup_{\tau_{h'} \in \text{hpe}(h)} ECB_{h'} \right) \cap UCB_k \right| \right\} \\ \gamma_{3,1}^{ecbu} &= \max_{\tau_k \in \{2,3\}} \left\{ \left| (ECB_1) \cap UCB_k \right| \right\} = \max \left\{ \left| ECB_1 \cap UCB_2 \right|, \left| ECB_1 \cap UCB_3 \right| \right\} = 4 \\ \gamma_{3,2}^{ecbu} &= \max \left\{ \left| (ECB_1 \cup ECB_2) \cap UCB_3 \right| \right\} = 6 \\ \gamma_{3,1}^{ecbu} + \gamma_{3,2}^{ecbu} &= 4 + 6 = 10\end{aligned}$$

Similarly to UCB-union based approaches, ECB-union based approaches compute CRPD upper bound of 10 reloads, thus approximating two cache block reloads over the safe bound.

Even when the lowest bound of the two is selected, CRPD bound is over-approximated by accounting for two cache block reloads which cannot occur in a single combination of preemptions during runtime. CRPD over-approximation is further increased when multiple jobs of each task are introduced. In this paper, we propose a novel method for computing the CRPD and the worst-case response time, accounting for the above-described problems.

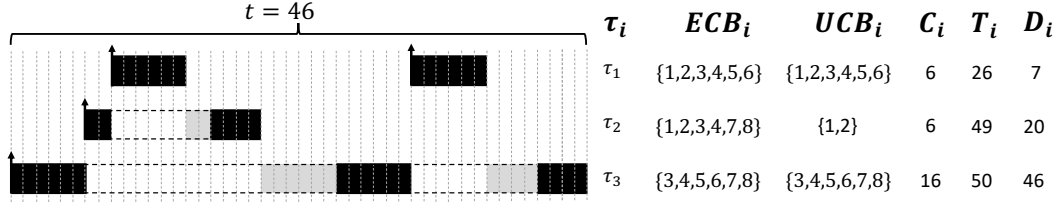
## 5 CRPD-aware Response-Time Analysis

In the remainder of the paper, when we refer to the term *preemption* we consider both, indirect (nested) and direct preemptions. We start with defining a cache-aware worst-case response time equation, slightly different than the existing ones. Formally, the response time analysis is defined as the least fixed-point of the following equation:

$$R_i^{(l+1)} = C_i + \gamma(i, R_i^{(l)}) + \sum_{\tau_h \in \text{hp}(i)} \left\lceil \frac{R_i^{(l)}}{T_h} \right\rceil C_h \quad (9)$$

Notice that unlike in the existing approaches, Equation 9 computes the CRPD upper bound  $\gamma(i, t)$ , which is a function that implicitly accounts for all preemptions that can occur within duration  $t$ , between the first  $i$  tasks of  $\Gamma$ . A CRPD upper bound on all preemptions that can occur within duration  $t$  can be computed more accurately by applying the following four steps that we describe in more detail in the remainder of this section:

1. Derive all possible preemptions which can occur within duration  $t$ , between the jobs of the first  $i$  tasks of  $\Gamma$ , (described in Subsection 5.1).
2. Divide the possible preemptions into partitions such that each partition accounts for single-job preemptions between the tasks, (described in Subsection 5.2)
3. Compute the CRPD bounds for each partition individually, (described in Subsection 5.3).
4. Sum the CRPD bounds of all partitions to obtain the cumulative CRPD bound on all possible preemptions within duration  $t$ , (described in Subsection 5.4).



■ **Figure 2** Worst-case preemptions for  $\tau_3$  during the time duration  $t = 46$ .

To show an overview of how the proposed analysis works, we provide the running example from Figure 2, and we compute the upper bound on CRPD within the time duration  $t = 46$ . The analysis computes the bound as follows:

1. Derive all possible preemptions which can occur within duration  $t$ , between the jobs of the first  $i$  tasks of  $\Gamma$ .

*Example:* Given the tasks from Figure 2, during the 46 time units, task  $\tau_1$  can preempt  $\tau_3$  at most two times, and it can preempt  $\tau_2$  at most once. Also,  $\tau_2$  can preempt  $\tau_3$  at most once. More formally, a single preemption from a job of  $\tau_h$  on a job of  $\tau_j$  is represented with an ordered pair  $(\tau_h, \tau_j)$ . Thus, all possible preemptions, within 46 time units, can be represented by the following multiset of ordered preemption pairs:

$$\{(\tau_1, \tau_3), (\tau_1, \tau_3), (\tau_1, \tau_2), (\tau_2, \tau_3)\} \quad (10)$$

2. Divide the possible preemptions into partitions such that each partition accounts for single-job preemptions between the tasks.

*Example:* To represent the partitions, we generate the multiset  $\Lambda$  which consists of all possible preemptions, divided into partitions that account for single-job preemptions between the tasks. Given the possible preemptions from the multiset derived in the previous step, the multiset  $\Lambda$  of all partitions is:

$$\Lambda = \left\{ \begin{array}{cc} \text{Partition 1} & \text{Partition 2} \\ \downarrow & \downarrow \\ \{(\tau_1, \tau_2), (\tau_1, \tau_3), (\tau_2, \tau_3)\} & , \quad \{(\tau_1, \tau_3)\} \end{array} \right\}$$

The multiset  $\Lambda$  consists of two partitions (each represented as a set of preemptions), such that the first partition consists of the following preemptions  $\{(\tau_1, \tau_2), (\tau_1, \tau_3), (\tau_2, \tau_3)\}$ , meaning that it is possible that  $\tau_1$  preempts  $\tau_2$ , that  $\tau_1$  preempts  $\tau_3$ , and that  $\tau_2$  preempts  $\tau_3$ . Jointly, the preemptions consist of all possible preemptions among the three tasks within a duration of 46 time units.

3. Compute the CRPD bounds for each partition individually.

*Example:* As we showed in the previous section, when a single job of each task may preempt the other jobs with lower priority, the upper bound on CRPD is 8 time units. This is the upper bound on all preemptions accounted in *Partition 1*. Considering *Partition 2*, it consists of a single preemption, from a job of  $\tau_1$  on  $\tau_3$ , and in this case the upper bound is 4 time units since the preemption may lead to the reloads of cache blocks 3, 4, 5 and 6.

4. Sum the CRPD bounds of all partitions to obtain the cumulative CRPD bound on all possible preemptions within duration  $t$ .

*Example:* The sum of upper bounds for *Partition 1* and *Partition 2* is  $8 + 4 = 12$ , which is the upper bound on all preemptions within 46 time units of the three shown tasks.

In the remainder of this section, we formally define the introduced terms, and prove that the proposed analysis results in a safe CRPD upper bound. The running example remains and it serves for better understanding on how the above values are computed and what they formally represent. This section is divided into the following subsections: 5.1 – describes the computation of upper bounds on the number of preemptions, 5.2 – describes the preemption partitioning, 5.3 – computation of CRPD bound for single partition, 5.4 – computation of CRPD bound for all preemptions, 5.5 – correctness proof for the computation of the worst-case response time, 5.6 – time complexity, and 5.7 – the additional computation for CRPD bound for single partition, based on finding the worst-case preemption combination.

### 5.1 Computing the upper bounds on the number of preemptions

► **Definition 1.** An upper bound  $E_j^h(t)$  on times a task  $\tau_h$  can preempt  $\tau_j$  ( $h < j$ ) within duration  $t$  is defined with the following equation:

$$E_j^h(t) = \begin{cases} \left\lceil \frac{t}{T_h} \right\rceil & , \left\lceil \frac{t}{T_h} \right\rceil \leq \left\lceil \frac{t}{T_j} \right\rceil \\ \left\lceil \frac{t}{T_j} \right\rceil \times \left\lceil \frac{R_j}{T_h} \right\rceil & , \left\lceil \frac{t}{T_h} \right\rceil > \left\lceil \frac{t}{T_j} \right\rceil \end{cases} \quad (11)$$

► **Proposition 2.**  $E_j^h(t)$  is an upper bound on number of possible preemptions from  $\tau_h$  on  $\tau_j$  within duration  $t$ .

**Proof.** Let us consider the following cases:

$\left\lceil \frac{t}{T_h} \right\rceil \leq \left\lceil \frac{t}{T_j} \right\rceil$ : Each job of  $\tau_h$  can preempt  $\tau_j$  at most once, therefore the number of  $\tau_h$  jobs which can be released within duration  $t$  is a safe bound on the number of preemptions from  $\tau_h$  on  $\tau_j$  within  $t$ .

$\left\lceil \frac{t}{T_h} \right\rceil > \left\lceil \frac{t}{T_j} \right\rceil$ : An upper bound on preemptions from jobs of  $\tau_h$  on a single job of  $\tau_j$  is equal to  $\left\lceil \frac{R_j}{T_h} \right\rceil$  since it is also an upper bound on number of times that jobs of  $\tau_h$  can be released within the worst-case response time  $R_j$  of a single job. Since Equation 11 applies the bound  $\left\lceil \frac{R_j}{T_h} \right\rceil$  on each job of  $\tau_j$  which can be released within  $t$ , the proposition holds. ◀

### 5.2 Preemption partitioning

Once the all possible preemptions which can occur within duration  $t$  are identified, we divide them into partitions, such that no partition accounts for the same preemption pair of the first  $i$  tasks in  $\Gamma$ , and such that all partitions jointly account for all possible preemptions.

► **Definition 3.** A multiset  $\Lambda_{i,t}$  of partitions consisting of all possible preemptions that can occur within duration  $t$ , between the jobs of the first  $i$  tasks of  $\Gamma$ .

$$\Lambda_{i,t} = \{\lambda_1, \lambda_2, \dots, \lambda_z\} \text{ such that } \lambda_r = \{(\tau_h, \tau_j) \mid r \leq E_j^h(t)\} \quad (12)$$

In Equation 12,  $\Lambda_{i,t}$  is defined as a multiset of sets (partitions). Each set  $\lambda_r$  consists of possible preemptions and each preemption is represented as an ordered pair  $(\tau_h, \tau_j)$  where the first element represents the preempting, and the second element represents the preempted task. The multiset  $\Lambda$  is formed of exactly  $z$  partitions, where  $z = \max\{E_j^h(t) \mid 1 \leq h < j \leq i\}$ . Each partition consists of disjunct preemptions, meaning that no partition contains two same preemption pairs.

*Example:* Given the taskset from the running example in Figure 2, the multiset  $\Lambda_{3,46}$  is computed as follows.

$$\Lambda_{3,46} = \{\lambda_1, \lambda_2\} \text{ where } \lambda_1 = \{(\tau_1, \tau_2), (\tau_1, \tau_3), (\tau_2, \tau_3)\} \text{ and } \lambda_2 = \{(\tau_1, \tau_3)\} \quad (13)$$



It is important to notice that the multiset union ( $\uplus$ ) of all partitions in  $\Lambda_{3,46}$  results in the multiset of all possible preemptions, e.g.,

$$\lambda_1 \uplus \lambda_2 = \{(\tau_1, \tau_2), (\tau_1, \tau_3), (\tau_2, \tau_3)\} \uplus \{(\tau_1, \tau_3)\} = \{(\tau_1, \tau_2), (\tau_1, \tau_3), (\tau_2, \tau_3), (\tau_1, \tau_3)\}$$

► **Proposition 4.** *Multiset  $\Lambda_{i,t}$  consists of all possible preemptions that may occur within duration  $t$ , between the jobs of the first  $i$  tasks of  $\Gamma$ .*

**Proof.** Directly follows from Proposition 2 and Equation 12 since Equation 12 includes each possible preemption, occurable within duration  $t$  between the first  $i$  tasks of  $\Gamma$ , in one of the partitions of  $\Lambda_{i,t}$ . ◀

### 5.3 CRPD bound on preemptions from a single partition

As suggested in Section 3, considering the *Problem 1* of CRPD over-approximation, computing a bound for different preemption partitions individually, instead of computing it for all preemptions at once, may result in more precise CRPD estimations.

To achieve this, once the multiset  $\Lambda_{i,t}$  of preemption partitions is computed, an upper bound on CRPD resulting from preemptions of a single partition  $\lambda_r \in \Lambda_{i,t}$  can be computed by selecting the minimum CRPD bound among the results from UCB-Union and ECB-Union approaches. Here, we describe the improvements and adjustments on those approaches to compute CRPD bound from preemptions contained within a partition.

In the following equations,  $\text{aff}(i, h, \lambda_r)$  represents a set of tasks with priorities higher than or equal to  $\tau_i$  and lower than  $\tau_h$  which can be preempted by  $\tau_h$  according to preemptions represented in  $\lambda_r$ . Formally:  $\text{aff}(i, h, \lambda_r) = \{\tau_k \mid (\tau_h, \tau_k) \in \lambda_r \wedge \tau_k \in \text{hpe}(i)\}$ . Also, with  $\text{hp}(i, \lambda_r)$  we denote a set of tasks with priorities higher than  $\tau_i$  such that for each  $\tau_h \in \text{hp}(i, \lambda_r)$  there is  $(\tau_h, \tau_i) \in \lambda_r$ .

First, we improve and adjust the ECB-Union approach, proposed by Altmeyer et al. [4]. In that approach, for a job of  $\tau_h$ , it is accounted that it directly preempts one of the tasks from  $\text{aff}(i, h, \lambda_r)$  set such that the maximum possible number of UCBs are evicted. In order for this approach to be correct, it is also accounted that tasks that can preempt  $\tau_h$  also contribute to the evictions of useful cache blocks of the preempted task. This scenario is represented by a CRPD bound  $\gamma_{i,h}^{\text{ecbp}}$ . We further improve this formulation by accounting that a preemption from a single job of  $\tau_h$  on any job of  $\tau_k$  from  $\text{aff}(i, h, \lambda_r)$  cannot cause more cache-block reloads than the maximum number of UCBs that can be evicted at a single preemption point of  $\tau_k$ . The maximum number of UCBs at a single preemption point of  $\tau_k$  is represented by  $\text{ucb}_k^{\text{max}}$ . The above translates to Equation 14.

$$\gamma_{i,h}^{\text{ecbp}}(\lambda_r) = \max_{\tau_k \in \text{aff}(i, h, \lambda_r)} \left\{ \min \left( \left| \left( \bigcup_{\tau_{h'} \in \text{hp}(h, \lambda_r) \cup \{\tau_h\}} \text{ECB}_{h'} \right) \cap \text{UCB}_k \right|, \text{ucb}_k^{\text{max}} \right) \right\} \quad (14)$$

We build the correctness of the proposed computation on the correctness of the standard ECB-Union method [4].

► **Proposition 5.**  $\gamma_{i,h}^{\text{ecbp}}(\lambda_r)$  is an upper bound on number of reloads that may be imposed by a direct preemption from  $\tau_h$  on one of the tasks within  $\text{aff}(i, h, \lambda_r)$  set.

**Proof.** A direct preemption from  $\tau_h$  on one of the tasks within  $\text{aff}(i, h, \lambda_r)$  set cannot cause more reloads than the maximum number of UCBs of a preemptable task, which can be evicted by  $\tau_h$  and all the tasks that may preempt  $\tau_h$ . Also, such bound cannot be greater than the maximum number of useful cache blocks  $\text{ucb}_k^{\text{max}}$  that may be present at a single preemption point within a preemptable task, which concludes the proof. ◀

The ECB-Union based upper bound on all preemptions from  $\lambda_r$  is computed by summing all  $\gamma_{i,h}^{ecbp}$  terms for each possibly preempting task, from  $\tau_1$  to  $\tau_{i-1}$ :

$$\gamma_i^{ecbp}(\lambda_r) = \sum_{h=1}^{i-1} \gamma_{i,h}^{ecbp}(\lambda_r) \quad (15)$$

► **Proposition 6.**  $\gamma_i^{ecbp}(\lambda_r)$  is an upper bound on number of reloads that can be caused by preemptions from the partition  $\lambda_r$ .

**Proof.** For each direct preemption from the preempting jobs of tasks from  $\tau_1$  to  $\tau_{i-1}$  in  $\lambda_r$ , Equation 15 accounts that the upper-bounded number of cache-blocks is reloaded in one of the preemptable jobs, as follows from Proposition 5. Therefore, the proposition holds. ◀

Next, we adjust the UCB-Union approach, proposed by Tan and Mooney [30]. In this approach, for a job of  $\tau_h$ , it is assumed that it can evict useful cache blocks from each task  $\tau_k$  from the  $\text{aff}(i, h, \lambda_r)$  set. However, since a single job of  $\tau_h$  can directly or indirectly preempt each  $\tau_k$  at only one of its preemption points, this cost can at most be equal to the sum of the number of maximum useful cache blocks at single preemption point of each task  $\tau_k$  from  $\text{aff}(i, h, \lambda_r)$ . The above is formally represented with  $\gamma_{i,h}^{ucbp}$  in the following equation:

$$\gamma_{i,h}^{ucbp}(\lambda_r) = \min \left( \left| \left( \bigcup_{\tau_k \in \text{aff}(i, h, \lambda_r)} UCB_k \right) \cap ECB_h \right|, \sum_{\tau_k \in \text{aff}(i, h, \lambda_r)} ucb_k^{max} \right) \quad (16)$$

We build the correctness of the proposed computation on the correctness of the standard UCB-Union method [30].

► **Proposition 7.**  $\gamma_{i,h}^{ucbp}(\lambda_r)$  is an upper bound on number of reloads within all tasks from  $\text{aff}(i, h, \lambda_r)$ , that may be imposed because of the cache-block accesses from a single job of  $\tau_h$ .

**Proof.** A job of  $\tau_h$  cannot impose more than one cache block reload per cache-memory block  $m$ , such that  $m \in ECB_h$ , and  $m \in UCB_k$  for any  $\tau_k$  such that  $\tau_k \in \text{aff}(i, h, \lambda_r)$ , as follows from UCB-Union [30]. Also, since each task  $\tau_k$  from  $\text{aff}(i, h, \lambda_r)$  can be preempted by  $\tau_h$  at only one of its preemption points, the maximum number of reloads from  $\tau_h$  cannot be greater than the sum of the maximum numbers of useful cache blocks that may be present at a preemption point within each such task. This concludes the proof. ◀

The UCB-Union based upper bound on all preemptions from  $\lambda_r$  is also computed by summing all  $\gamma_{i,h}^{ucbp}$  terms for each possibly preempting task from  $\tau_1$  to  $\tau_{i-1}$ :

$$\gamma_i^{ucbp}(\lambda_r) = \sum_{h=1}^{i-1} \gamma_{i,h}^{ucbp}(\lambda_r) \quad (17)$$

► **Proposition 8.**  $\gamma_i^{ucbp}(\lambda_r)$  is an upper bound on number of reloads that can be caused by preemptions from the partition  $\lambda_r$ .

**Proof.** For each possibly preempting job from  $\tau_1$  to  $\tau_{i-1}$  in  $\lambda_r$ , Equation 17 accounts that the job leads to upper-bounded number of cache-block reloads in its possibly preemptable jobs, as follows from Proposition 7. Therefore, the proposition holds. ◀

The final upper bound  $\gamma_i(\lambda_r)$  on CRPD from possible preemptions given in  $\lambda_r$ , between single jobs of the first  $i$  tasks from  $\Gamma$ , is defined as the least bound of the two.

$$\gamma_i(\lambda_r) = \min \left( \gamma_i^{ecbp}(\lambda_r), \gamma_i^{ucbp}(\lambda_r) \right) \times BRT \quad (18)$$

► **Proposition 9.**  $\gamma_i(\lambda_r)$  is an upper bound on CRPD from possible preemptions given in the partition  $\lambda_r$ .

**Proof.** Follows from Propositions 6 and 8 since Equation 18 results in the least bound. ◀

## 5.4 CRPD bound on all preemptions within a time interval

Now, we define a computation for the CRPD bound on all preemptions which can occur within duration  $t$ , between the first  $i$  tasks of  $\Gamma$ .

► **Definition 10.** An upper bound  $\gamma(i, t)$  on CRPD of all preemptions, which can occur within duration  $t$  between the first  $i$  tasks of  $\Gamma$ , is defined with the following equation:

$$\gamma(i, t) = \sum_{k=1}^{|\Lambda_{i,t}|} \gamma_i(\lambda_r) \quad (19)$$

► **Proposition 11.**  $\gamma(i, t)$  is an upper bound on CRPD of all preemptions which can occur within duration  $t$  between the first  $i$  tasks of  $\Gamma$ .

**Proof.** Directly follows from Propositions 4 and 9, since Equation 19 is a sum of CRPD upper bounds of preemption partitions that jointly consist of all preemptions within  $t$ . ◀

## 5.5 Worst-case response time

In this subsection, we prove that the computed worst-case response time is an upper bound.

► **Theorem 12.**  $R_i$  is an upper bound on worst-case response time of  $\tau_i$ .

**Proof.** By induction, over the tasks in  $\Gamma$  in a decreasing priority order.

*Base case:*  $R_1 = C_1$ , because  $hp(i) = \emptyset$ . Since  $C_i$  is the worst-case execution time of  $\tau_1$  the proposition holds.

*Inductive hypothesis:* Assume that for all  $\tau_h \in hp(i)$ ,  $R_h$  is an upper bound on worst-case response time of  $\tau_h$ .

*Inductive step:* We show that Equation 9 computes the worst-case response time of  $\tau_i$ . Consider the least fixed point of Equation 9, for which  $R_i = R_i^{(l)} = R_i^{(l+1)}$ . At this point, the equation accounts for the following upper bounds and worst-case execution times:

- ◊  $C_i$ , which is the worst-case execution time of  $\tau_i$ , assumed by the system model.
- ◊  $\gamma(i, R_i)$ , which is proved by Proposition 11 to be an upper bound on CRPD of all jobs which can be released within duration  $R_i$ , and have higher than or equal priority to  $\tau_i$ .
- ◊  $\sum_{\tau_h \in hp(i)} \lceil R_i / T_h \rceil C_h$ , which is the worst-case interference caused by execution of all jobs of tasks with higher priority than  $\tau_i$  without CRPD. Since we proved for all the factors which can prolong the response time of  $\tau_i$  that they are accounted as the respective execution and CRPD upper bounds in Equation 9, then their sum results in an upper bound. ◀

## 5.6 Time complexity

The time complexity of the proposed analysis can be improved since in its current form Equation 12 explicitly creates all partitions which can lead to re-computation of CRPD bounds for many identical partitions. For this reason, we first define the matrix  $A_{i,t}$ , from which it is possible to identify how many repeated partitions there are, and compute CRPD bound for each distinct partition only once, as introduced in Algorithm 1.

► **Definition 13.** A matrix  $A_{i,t}$  of upper bounds on number of preemptions between each pair of tasks with higher than or equal priority to  $P_i$  which can occur within duration of  $t$ , is defined with the following equation:

$$A_{i,t} = (a_{j,h}) \in \mathbb{N}^{i \times i} \mid a_{j,h} = \begin{cases} 0 & , j \leq h \\ E_j^h(t) & , j > h \end{cases} \quad (20)$$

► **Proposition 14.**  $A_{i,t}$  stores an upper-bounded number of preemptions, which can occur within  $t$ , between each pair of tasks with higher than or equal priority to  $P_i$ .

**Proof.** Proposition 14 follows directly from Proposition 2 and the fact that  $\tau_j$  cannot preempt any task  $\tau_h$  of higher priority, or  $\tau_j$  itself ( $j \leq h$ ). ◀

Equation 20 defines a square matrix  $A_{i,t}$  such that the number of rows and columns is equal to  $i$ , and each entry of the matrix represents the maximum number of preemptions from a task  $\tau_h$  on  $\tau_j$  within duration  $t$ .

*Example:* Given the taskset from Figure 2, a matrix of preemptions during 46 time units looks as follows:

$$A_{3,46} = \begin{array}{ccc|c} \tau_1 & \tau_2 & \tau_3 & \\ \hline 0 & 0 & 0 & \tau_1 \\ 1 & 0 & 0 & \tau_2 \\ 2 & 1 & 0 & \tau_3 \end{array} \quad (21)$$

The element  $a(2,1) = 1$  represents the maximum number of preemptions from  $\tau_1$  on  $\tau_2$  during 46 time units (note  $R_2 = 14$ ).

*Algorithm explanation:* In a matrix  $A_{i,t}$ , there are at most  $\frac{n*(n-1)}{2}$  values representing different numbers of possible preemptions among the tasks. Therefore, there are at most  $\frac{n*(n-1)}{2}$  distinct partitions to be generated. In Algorithm 1, we define the procedure that first generates  $A_{i,t}$  (line 3), and then generates distinct partitions one by one (lines 4 – 10). For each distinct partition, we compute the number  $sp$  of times a partition is repeated, then

■ **Algorithm 1** Algorithm for computing the cumulative CRPD during a time interval of length  $t$ .

---

**Data:** Time duration  $t$ , task index  $i$ , Taskset  $\Gamma$   
**Result:** CRPD upper bound  $\xi$  on all jobs with priority higher than or equal to  $P_i$ , which can be released within duration  $t$ .

---

```

1 fn  $\gamma(i, t)$ 
2    $\xi \leftarrow 0$ 
3    $A_{i,t} \leftarrow$  generate the matrix of maximum preemption counts between the tasks
   (Equation 20)
4   while  $A_{i,t} \neq 0_{i \times i}$  do
5      $sp \leftarrow$  minimum value from  $A_{i,t}$ , greater than zero
6      $\lambda \leftarrow \{(\tau_h, \tau_j) \mid sp \leq a_{j,h}\}$ 
7      $\gamma_i(\lambda) \leftarrow$  compute the CRPD upper bound from the preemptions in  $\lambda$ 
8      $\xi \leftarrow \xi + sp \times \gamma_i(\lambda)$ 
9      $A_{i,t} \leftarrow$  decrease all values, greater than zero, by  $sp$ 
10  end
11  return  $\xi$ 

```

---

compute the partition (line 6), and account for its CRPD bound  $sp$  times in the cumulative CRPD bound  $\xi$  that is updated for each distinct partition (lines 7 and 8). After this, the partitioned preemptions are removed (line 9), and the next distinct partition is computed until no more preemptions are left to be partitioned. Formally, termination criteria is satisfied when  $A_{i,t}$  equals to the zero matrix  $0_{i \times i}$ . At the end, the algorithm results in the same CRPD bound as Equation 19. Using this algorithm, the time complexity is  $\mathcal{O}(n^3 * x)$ , where the complexity of computation at line 6 is  $\mathcal{O}(x)$ .

## 5.7 CRPD computation using preemption scenarios

In this section, we propose an alternative computation for the upper bound  $\gamma_i(\lambda)$  on CRPD of preemptions in the partition  $\lambda$ . The goal is to compute the CRPD bound from a single worst-case preemption combination among the preemptions from  $\lambda$ , addressing *Problem 2* from Section 4. To achieve this, we first formally define the following terms:

- *Preemption scenario*  $(\tau_i, PT)$ , and its CRPD upper bound  $\gamma(\tau_i, PT)$ ,
- *Preemption combination*  $\Pi_\lambda^c$ , and its CRPD upper bound  $\gamma(\Pi_\lambda^c)$ .

Informally, we define a preemption combination as a set of feasible preemptions where only one job of each task is involved, such that all accounted preemptions are present in a partition  $\lambda$ . Before being able to formally define a preemption combination, we first formally define a preemption scenario.

► **Definition 15** (Preemption scenario  $(\tau_i, PT)$ ). *A preemption scenario represents a single interruption due to preemption of a task and it is defined as an ordered pair  $(\tau_i, PT)$ , where  $\tau_i$  represents the preempted (interrupted) task, and  $PT$  is a set of preempting tasks which execute after the interruption at  $\tau_i$  and before the immediate resumption of  $\tau_i$ . Formally, for each preemption scenario  $(\tau_i, PT)$  it holds that  $PT \subseteq hp(i)$ .*

*Example:* Given the example from Fig. 2, the first preemption scenario in  $\tau_3$  is  $(\tau_3, \{\tau_1, \tau_2\})$ , and the second preemption scenario is  $(\tau_3, \{\tau_1\})$ . Also, in the same figure,  $\tau_2$  is preempted once and this preemption scenario is  $(\tau_2, \{\tau_1\})$ .

In order to compute the upper bound on CRPD on  $\tau_i$ , resulting from one interruption scenario, the ordering of the preempting tasks is not important. All of them are equally capable of evicting cache blocks of  $\tau_i$  between its preemption and resumption, regardless of their ordering.

► **Definition 16** (CRPD of a preemption scenario  $\gamma(\tau_i, PT)$ ). *An upper bound  $\gamma(\tau_i, PT)$  on the CRPD of a preempted task  $\tau_i$  resulting from a preemption scenario  $(\tau_i, PT)$  is:*

$$\gamma(\tau_i, PT) = |UCB_i \cap \bigcup_{\tau_h \in PT} ECB_h| \times BRT \quad (22)$$

► **Proposition 17.**  *$\gamma(\tau_i, PT)$  is an upper bound on the CRPD of a preempted task  $\tau_i$  resulting from a preemption scenario  $(\tau_i, PT)$ .*

**Proof.** Since Equation 22 accounts that each UCB from  $\tau_i$  is definitely reloaded with the worst-case block reload time if there is a corresponding evicting cache block from any of the preempting tasks within a scenario, the proposition holds. ◀

*Example:* Given the preemption scenario  $(\tau_3, \{\tau_1, \tau_2\})$ , the upper bound on CRPD of  $\tau_3$  is:

$$\gamma(\tau_3, \{\tau_1, \tau_2\}) = |UCB_3 \cap \{ECB_1 \cup ECB_2\}| = 6$$

A safe upper bound on CRPD of  $\tau_i$  resulting from a preemption scenario  $(\tau_i, PT)$  can be tightened even more if the low-level task analysis can provide more detailed information on UCBs and ECBs at different program points. In such case, the bound is computed as the maximum intersection of UCBs from a single point within  $\tau_i$  and the evicting cache blocks of tasks in  $PT$ . This is the case because Equation 22 considers a single preempted point and tasks which may evict cache blocks before preempted task resumes to execute. On the other hand, many existing approaches consider multiple preemption scenarios at once, and therefore this improvement is not applicable in their case, as shown by Shah et al. [27].

Given the formal definition of preemption scenarios, now we can define a preemption combination. With this definition, we need to insure that a preemption combination consists only of preemption scenarios which account for interactions between a single job of each task. Therefore, we need to insure that a preemption combination does not include two preemption scenarios which are mutually exclusive, given the constraint of using only single jobs.

► **Definition 18** (Preemption combination  $\Pi^c$ ). *A preemption combination  $\Pi^c$  is defined as a set of disjoint non-empty preemption scenarios between single jobs of tasks in  $\Gamma$  such that:*

- 1) *If there are two preemption scenarios  $(\tau_j, PT^j)$  and  $(\tau_l, PT^l)$  in  $\Pi^c$  such that  $\tau_h \in PT^j \cap PT^l$  and  $P_l < P_j$ , it implies that  $\tau_j \in PT^l$ .*
- 2) *Each preempting task  $\tau_h \in PT$ , where  $(\tau_j, PT) \in \Pi^c$ , can be in at most one preemption scenario imposed on  $\tau_j$ .*

**The first constraint** refers to a case: If a single job of task  $\tau_h$  preempts single jobs of tasks  $\tau_j$  and  $\tau_l$ , where  $P_j > P_l$ , then that job of  $\tau_l$  is definitely preempted by the  $\tau_j$  job.

**The second constraint** accounts that an additional preemption scenario with  $\tau_h$  implies that  $\Pi^c$  accounted for two jobs of  $\tau_h$  preempting a job of  $\tau_j$ , while the definition accounts for at most one job of each task.

*Example:* Given a definition of a preemption combination, one possible combination is:  $\{(\tau_3, \{\tau_1\}), (\tau_3, \{\tau_2\})\}$  which describes the preemption scenario where  $\tau_1$  directly preempts  $\tau_3$  at one preemption point, while  $\tau_2$  directly preempts  $\tau_3$  at another preemption point. However, the set  $\{(\tau_3, \{\tau_1\}), (\tau_2, \{\tau_1\})\}$  is not a preemption combination, because it describes the case where a job of  $\tau_3$  is preempted by a job of  $\tau_1$ , and a job of  $\tau_2$  is preempted by a job of  $\tau_1$ , while a job of  $\tau_2$  does not preempt a job of  $\tau_3$ . Since this is the case, more than two jobs of  $\tau_1$  are accounted, which violates Definition 18.

► **Definition 19** (Preemption combination consistent with  $\lambda$ ). *We say that  $\Pi^c$  is a preemption combination consistent with the preemption partition  $\lambda$  iff for any preemption scenario  $(\tau_j, PT) \in \Pi^c$  the preemptions captured by the scenario are possible, i.e. present in  $\lambda$ . Formally:  $\forall (\tau_j, PT) \in \Pi_i^c : \forall \tau_h \in PT : (\tau_j, \tau_h) \in \lambda$ .*

*Example:* Given the preemption partition  $\lambda = \{(\tau_1, \tau_2), (\tau_1, \tau_3), (\tau_2, \tau_3)\}$ , a preemption combination consistent with  $\lambda$  is  $\{(\tau_3, \{\tau_1\}), (\tau_3, \{\tau_2\})\}$  since it describes preemption scenarios made of preemptions that are possible, i.e. present in  $\lambda$ .

► **Definition 20** (CRPD  $\gamma(\Pi^c)$  of a preemption combination). *An upper bound  $\gamma(\Pi^c)$  on the CRPD of a single preemption combination  $\Pi^c$  is defined as the sum of upper bounds of all preemption scenarios in  $\Pi^c$ . Formally, it is defined as:*

$$\gamma(\Pi^c) = \sum_{(\tau_k, PT) \in \Pi^c} \gamma(\tau_k, PT) \quad (23)$$

► **Proposition 21.**  *$\gamma(\Pi^c)$  is an upper bound on CRPD of preemptions accounted within  $\Pi^c$ .*

**Proof.** A combination consists of a number of preemption scenarios representing the preemptions from preempting tasks on different preemption points. Following from Proposition 17, a sum of CRPD upper bounds of each task interruption in a combination is an upper bound on CRPD of all preemptions accounted in a combination, which concludes the proof. ◀

*Example:* Given the preemption combination of two direct preemptions from Figure 1, CRPD upper bound is:  $\gamma(\{(\tau_3, \{\tau_1\}), (\tau_3, \{\tau_2\})\}) = \gamma(\tau_3, \{\tau_1\}) + \gamma(\tau_3, \{\tau_2\}) = 4 + 4 = 8$ . Given the preemption combination of a nested preemption from the figure, it is:  $\gamma(\{(\tau_3, \{\tau_1, \tau_2\}), (\tau_2, \{\tau_1\})\}) = \gamma(\tau_3, \{\tau_1, \tau_2\}) + \gamma(\tau_2, \{\tau_1\}) = 6 + 2 = 8$ .

Now, we can imagine a set which consists of all possible preemption combinations which are consistent with preemptions enlisted in  $\lambda$ . Then, among all the generated preemption combinations, we find one which results in the worst-case CRPD and declare that as a safe upper-bound, since that is the maximum obtainable CRPD value among all the possible preemption combinations.

However, to generate such complete set of all possible preemption combinations is computationally inefficient. A potential solution can come from the fact that it is enough to compute a subset of the complete set of combinations, as long as we are sure that no greater CRPD value can be obtained in the remaining, unaccounted combinations. We show this with the following *example*: Given a set of possible preemptions  $\lambda = \{(\tau_1, \tau_2), (\tau_1, \tau_3), (\tau_2, \tau_3)\}$  from Figure 1, let us consider the following set of two combinations:

$$\Pi_{3,\lambda} = \left\{ \left\{ (\tau_3, \{\tau_1\}), (\tau_3, \{\tau_2\}) \right\}, \left\{ (\tau_3, \{\tau_1, \tau_2\}), (\tau_2, \{\tau_1\}) \right\} \right\}$$

$\Pi_{3,\lambda}$  consists of: 1) a combination of direct preemption scenarios, and 2) a combination of nested preemption. Any other possible preemption combination, from preemptions in  $\lambda$ , can only be derived by omitting at least one preemption from a preemption scenario from one of the two combinations given in  $\Pi_{3,\lambda}$ . E.g. preemption combination  $\{(\tau_3, \{\tau_1, \tau_2\})\}$  is equal to and results in the same CRPD as  $\{(\tau_3, \{\tau_1, \tau_2\}), (\tau_2, \emptyset)\}$ . Also, all of the preemption scenarios from  $\{(\tau_3, \{\tau_1, \tau_2\})\}$  are already included in the second combination. Therefore, all the other possible combinations cannot result in a greater CRPD value than those in  $\Pi_{3,\lambda}$ , meaning that this subset is sufficient to compute a safe CRPD.

A preemption combination which is constructed by adding a preemption scenario to any of the combinations in  $\Pi_{3,\lambda}$  cannot be obtained. This is the case because in the first combination, it is accounted that  $\tau_3$  is preempted by both tasks, at two different points, accounting for all preemptions in  $\lambda$  where  $\tau_3$  can be preempted, i.e.  $(\tau_1, \tau_3)$  and  $(\tau_2, \tau_3)$ . In this case,  $\tau_2$  cannot be preempted considering preemption  $(\tau_1, \tau_2) \in \lambda$  as shown in Definition 18. In the second combination,  $\tau_3$  is interrupted once while both tasks preempt it, and  $\tau_2$  is preempted by  $\tau_1$ , meaning that all preemptions from  $\lambda$  are accounted.

In order to define a safe set of combinations  $\Pi_{i,\lambda}$ , such that at least one of those combinations may result in the worst-case CRPD, we first introduce a term of **set partitioning**<sup>1</sup> in order to represent different ways one task may be preempted by the others. *Example:* Given the task  $\tau_3$ , set partitions of a set  $\{\tau_1, \tau_2\}$  of its potentially preempting tasks are: 1)  $\{\{\tau_1, \tau_2\}\}$ , and 2)  $\{\{\tau_1\}, \{\tau_2\}\}$ .

Given a preemptable task  $\tau_k$ , and a set of its possibly preempting tasks  $PT$ , all set partitions of  $PT$  represent all the ways  $\tau_k$  may be preempted such that each task from  $PT$  preempts  $\tau_k$ . This is the case because set partitions represent all the ways a set can

<sup>1</sup> Set partitioning is a mathematical concept sometimes also known as Bell partitioning [1, 18] named after Eric Temple Bell. There are many fast algorithms for generating set partitions, e.g. [13, 14].



be grouped in non-empty subsets, such that each set element is included in exactly one subset. Analogically, in this paper, each set partition is transformed into a preemption combination defining one way how a task (e.g.  $\tau_3$  above) can be preempted. Each set partition consists of subsets, and each subset represents a preemption scenario on the preempted task. This transformation is formally defined in function  $generateCombs(PT, \tau_k)$  in Algorithm 2. *Example:* Considering different ways  $\tau_3$  can be preempted, set partition  $\{\{\tau_1, \tau_2\}\}$  consists of a single subset, and forms a preemption combination  $\{(\tau_3, \{\tau_1, \tau_2\})\}$ , while set partition  $\{\{\tau_1\}, \{\tau_2\}\}$  consists of two subsets and forms a preemption combination  $\{(\tau_3, \{\tau_1\}), (\tau_3, \{\tau_2\})\}$  with two preemption scenarios:  $(\tau_3, \{\tau_1\})$ , and  $(\tau_3, \{\tau_2\})$ .

► **Proposition 22.** *Given a preemptable task  $\tau_k$ , and a set of possibly preempting tasks  $PT$ , any combination of preemptions on  $\tau_k$  will result in a less than or equal CRPD than any combination generated from  $generateCombs(PT, \tau_k)$ .*

**Proof.** By contradiction: Let us assume that there is a preemption combination  $\Pi_k^c$  representing the ways how  $\tau_k$  can be preempted by tasks from  $PT$ , and that  $\Pi_k^c$  can result in a greater CRPD than any combination derived from  $generateCombs(PT, \tau_k)$ . The combinations generated from  $generateCombs(PT, \tau_k)$  represent all the ways  $\tau_k$  may be preempted such that each task from  $PT$  preempts  $\tau_k$  since set partitions represent all the ways a set can be grouped in non-empty subsets, such that each element is included in exactly one subset. Thus,  $\Pi_k^c$  must omit at least one preemption, compared to at least one preemption combination derived from  $generateCombs(PT, \tau_k)$ . The initial assumption therefore contradicts Proposition 21 because  $\Pi_k^c$  cannot impose larger CRPD than the corresponding preemption combination from  $generateCombs(PT, \tau_k)$ , which accounts for the same preemptions as in preemption scenarios from  $\Pi_k^c$  and at least one additional preemption compared to  $\Pi_k^c$ . ◀

Using the concept of set partitioning to represent the ways a single task may be preempted, we generate a set  $\Pi_{i,\lambda}$  of preemption combinations on how all tasks from  $\tau_i$  to  $\tau_1$  can interact among each other:

► **Definition 23.** *By  $\Pi_{i,\lambda}$  we denote the result from Algorithm 2, i.e. the set of preemption combinations between the first  $i$  tasks of  $\Gamma$  such that each combination is consistent with  $\lambda$ .*

We describe Algorithm 2 in more details and we use a running example from Figure 2 to show the algorithm walk-through in Figure 3. As stated before, for each  $\tau_k$ , from  $\tau_i$  to  $\tau_1$ , the algorithm first generates possible combinations on how  $\tau_k$  can be preempted, using set partitioning (line 3). This process is defined in function  $generateCombs$  (line 7) and it translates the set partitions of possibly preempting tasks on  $\tau_k$ , into different ways  $\tau_k$  can be preempted, which is represented with a set of preemption combinations (line 16). Then, for each of those combinations, the algorithm performs  $extendCombs$  (line 4), which is a function that updates the existing preemption combinations, with further preemption scenarios that are possible on the preempting tasks of  $\tau_k$ . Take for example the preemption combination  $\Pi_k^c$ , given in Figure 3.

The combination represents the case where  $\tau_4$  is preempted at one preemption point, by all of its three possibly preempting tasks ( $\tau_1, \tau_2, \tau_3$ ). In the figure, this is represented by one arrow (standing for one preempted point of  $\tau_4$ ) and tasks preempting a point (above the arrow). Function  $extendCombs()$  further computes possible ways of preempting  $\tau_3$  since  $\tau_3$  is the lowest-priority preempting task from the preemption scenario  $(\tau_4, \{\tau_1, \tau_2, \tau_3\})$ . Those ways are represented with a set  $\Pi_3'$  of preemption combinations. After this, the function updates the preemption combinations with a Cartesian product of the two. Therefore, on the right side of the figure, you may notice that now we have two new combinations, updating the  $\Pi_k^c$  with different ways  $\tau_3$  can be preempted. The topmost combination can be updated further on, since preemption scenario  $(\tau_3, \{\tau_1, \tau_2\})$  can be updated with additional scenario on how  $\tau_2$  can be preempted by  $\tau_1$ . This is eventually computed within the algorithm because condition



in line 18 insures that all combinations are updated until no new preemption scenario can be added to any of the existing preemption combinations. More formally, this criteria is satisfied when for each preemption scenario  $(\tau_x, PT)$  within any preemption combination from  $\Pi_{i,\lambda}$ , function  $extended?((\tau_x, PT))$  yields true ( $\top$ ), meaning that all preemption scenarios are extended.

► **Proposition 24.**  $\Pi_{i,\lambda}$  is a safe set of preemption combinations between the single jobs of the first  $i$  tasks in  $\Gamma$ , i.e. there is no preemption combination consistent with  $\lambda$  with a higher CRPD than the maximum CRPD of the combinations in  $\Pi_{i,\lambda}$ ,

■ **Algorithm 2** Algorithm that generates a set  $\Pi_{i,\lambda}$  of preemption combinations.

---

**Data:** Set of possible preemption pairs  $\lambda$ , task index  $i$   
**Result:** A set  $\Pi_{i,\lambda}$  of preemption combinations consistent with  $\lambda$

```

1  $\Pi_{i,\lambda} \leftarrow \emptyset$ 
2 for  $k \leftarrow i$  to 2 by  $-1$  do
3    $\Pi'_{k,\lambda} \leftarrow generateCombs(hp(k), \tau_k)$ 
4    $\Pi_{i,\lambda} \leftarrow \Pi_{i,\lambda} \cup extendCombs(\Pi'_{k,\lambda})$ 
5 end
6 return  $\Pi_{i,\lambda}$ 
7 fn  $generateCombs(PT, \tau_k)$ 
8    $\Pi_{k,\lambda} \leftarrow \emptyset$ 
9    $PT_{k,\lambda} \leftarrow$  remove those tasks from  $PT$  that cannot preempt  $\tau_k$  according to  $\lambda$ 
10   $partitions(PT) \leftarrow$  generate all possible partitions of a set  $PT_{k,\lambda}$ , representing ways a job of  $\tau_k$  can be preempted.
11  for each  $partition \in partitions(PT)$ 
12     $\Pi_k^c \leftarrow \emptyset$ 
13    for each  $subset \in partition$ 
14       $\Pi_k^c \leftarrow \Pi_k^c \cup \{(\tau_k, subset)\}$ 
15     $\Pi_{k,\lambda} \leftarrow \Pi_{k,\lambda} \cup \Pi_k^c$ 
16  return  $\Pi_{k,\lambda}$ 
17 fn  $extendCombs(\Pi_{q,\lambda})$ 
18  while  $\exists \Pi^c \in \Pi_{q,\lambda} : \exists (\tau_r, PT) \in \Pi^c \mid extended?((\tau_r, PT), \Pi^c) = \perp$  do
19     $\tau_l \leftarrow$  lowest-priority task in  $PT$ 
20     $\Pi'_l \leftarrow generateCombs(PT \setminus \tau_l, \tau_l)$ 
21    for each  $\Pi^c \in \Pi_{q,\lambda} \mid (\tau_r, PT) \in \Pi^c$ 
22       $\Pi_{q,\lambda} \leftarrow \Pi_{q,\lambda} \cup (\Pi^c \times \Pi'_l)$ 
23       $\Pi_{q,\lambda} \leftarrow \Pi_{q,\lambda} \setminus \Pi^c$ 
24  end
25  return  $\Pi_{q,\lambda}$ 
26 fn  $extended?((\tau_r, PT), \Pi^c)$ 
27   $\tau_l \leftarrow$  lowest-priority task in  $PT$ 
28  if  $\exists (\tau_x, PT') \in \Pi^c \mid \tau_x = \tau_l$  then
29    return  $\top$ 
30  else return  $\perp$ ;

```

---

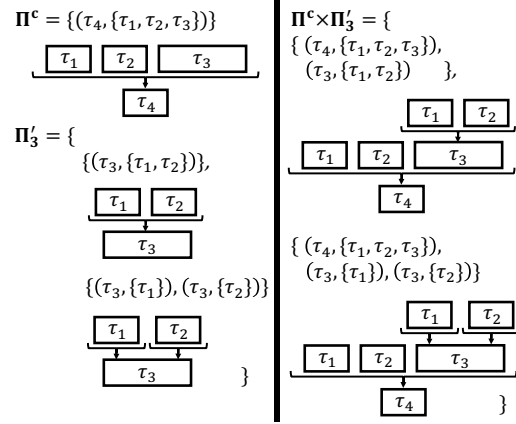
**Data:**  $\lambda = \{(\tau_1, \tau_2), (\tau_1, \tau_3), (\tau_2, \tau_3)\}$ ,  $i = 3$   
**Algorithm run:**

```

 $\Pi_{3,\lambda} \leftarrow \emptyset$ 
for  $k = 3$ 
   $\Pi'_{3,\lambda} \leftarrow generateCombs(hp(3), \tau_3)$ 
   $\leftarrow \{ \{(\tau_3, \{\tau_1\}), (\tau_3, \{\tau_2\})\}, \{(\tau_3, \{\tau_1, \tau_2\})\} \}$ 
   $\Pi_{3,\lambda} \leftarrow \emptyset \cup extendCombs(\Pi'_{3,\lambda})$ 
   $\leftarrow \{ \{(\tau_3, \{\tau_1\}), (\tau_3, \{\tau_2\})\}, \{(\tau_3, \{\tau_1, \tau_2\}), (\tau_2, \{\tau_1\})\} \}$ 
for  $k = 2$ 
   $\Pi'_{2,\lambda} \leftarrow generateCombs(hp(2), \tau_2)$ 
   $\leftarrow \{ \{(\tau_2, \{\tau_1\})\} \}$ 
   $\Pi_{3,\lambda} \leftarrow \{ \{(\tau_3, \{\tau_1\}), (\tau_3, \{\tau_2\})\}, \{(\tau_3, \{\tau_1, \tau_2\}), (\tau_2, \{\tau_1\})\} \} \cup \{ \{(\tau_2, \{\tau_1\})\} \}$ 
return  $\Pi_{3,\lambda}$ 

```

---



■ **Figure 3** Top: Algorithm walkthrough with an example from Figure 2. Bottom: Example for extending the combination  $\Pi^c$  of four tasks.

**Proof.** By contradiction: Let us assume that there is a preemption combination  $\Pi_o^c$  between the single jobs of the first  $i$  tasks, consistent with  $\lambda$ , which can result in a higher CRPD than any of the combinations in  $\Pi_{i,\lambda}$ . For each task  $\tau_k$  such that  $(1 < k \leq i)$ , it is accounted that  $\tau_k$  experiences the worst-case CRPD at one of the generated combinations, as follows from

Proposition 22 and line 3. Each such a combination is extended in line 4, accounting for further worst-case preemption scenarios on how all the preempting tasks can be preempted, and Algorithm 2 stops only when no more preemption scenarios can be generated and added to a set of preemption combinations  $\Pi_{i,\lambda}$ . This further implies that  $\Pi_o^c$  must omit at least one preemption from at least one of its preemption scenarios compared to any combination from  $\Pi_{i,\lambda}$ . Moreover, by construction of Algorithm 2, there is a preemption combination  $\Pi_w^c$  in  $\Pi_{i,\lambda}$  which is a superset over the  $\Pi_o^c$ , i.e. there is a mapping of preemption scenarios between  $\Pi_w^c$  and  $\Pi_o^c$  such that each preemption scenario of  $\Pi_w^c$  includes same preemptions as the respective scenario in  $\Pi_o^c$ , but may also include additional ones not accounted by  $\Pi_o^c$ . As follows from Propositions 21 and 22,  $\Pi_o^c$  can only result in CRPD less than or equal to the one from  $\Pi_w^c$ . This contradicts the initial assumption.  $\blacktriangleleft$

Finally, we can compute an upper bound on CRPD resulting from the worst-case preemption combination consisting of the preemptions in  $\lambda$ , with the following equation:

$$\gamma_i(\lambda) = \max_{\Pi^c \in \Pi_{i,\lambda}} \gamma(\Pi^c) \quad (24)$$

► **Proposition 25.**  $\gamma_i(\lambda)$  is an upper bound on CRPD from preemptions given in the partition  $\lambda$ , between the single jobs of the first  $i$  tasks from  $\Gamma$ .

**Proof.** Equation 24 computes the maximum upper bound from all preemption combinations accounted by  $\Pi_{i,\lambda}$ . Then, following from Propositions 21 and 24, the proposition holds.  $\blacktriangleleft$

*Example:* Given a set of possible preemptions  $\lambda = \{(\tau_1, \tau_2), (\tau_1, \tau_3), (\tau_2, \tau_3)\}$  from Figure 1 and continuing from the example after Proposition 21, the upper bound on CRPD resulting from preemptions in  $\lambda$  is computed as  $\gamma(\lambda) = \max(\{8, 8\}) = 8$ .

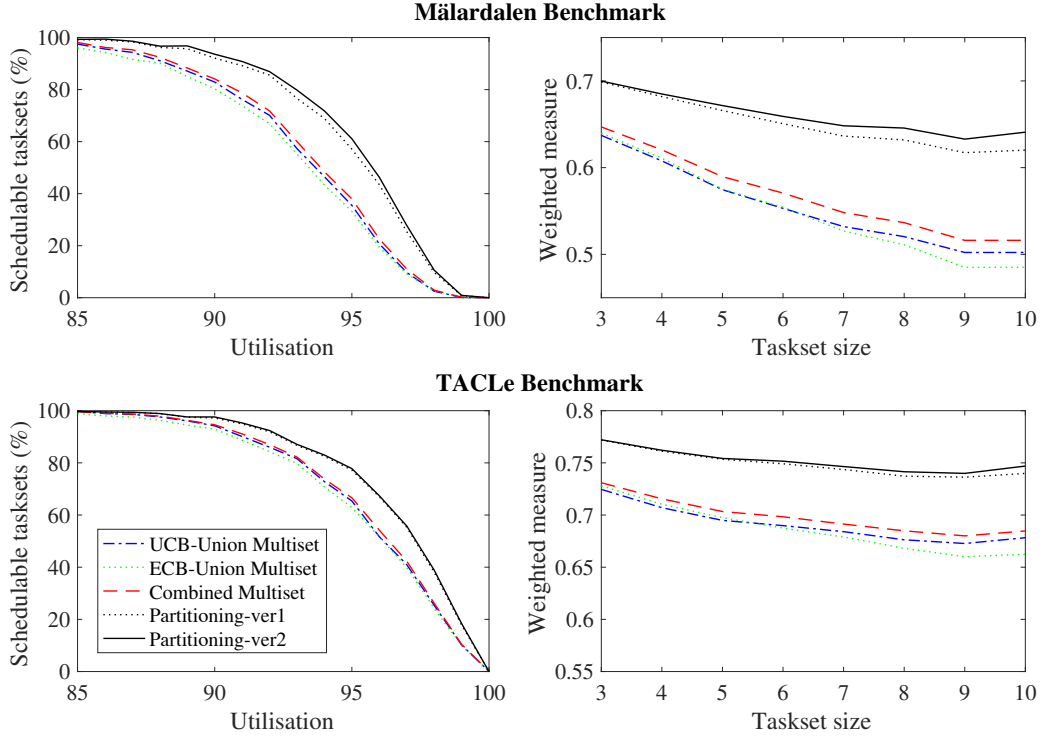
## 5.8 Adjustment for LRU caches

The proposed methods can also be used for set-associative LRU caches with a single modification, as shown by Altmeyer et al. [4, 6]. In case of LRU set-associative cache, a cache-set may contain several useful cache blocks, e.g.,  $UCB_2 = \{1, 2, 2, 2\}$  means that  $\tau_2$  contains three cache blocks in cache-set 2, and one UCB in cache set 1. Upon preemption, one ECB of the pre-empting task may suffice to evict all UCBs of the same cache-set, meaning that  $ECB_1 = \{1, 2\}$  may evict all cache blocks of  $\tau_2$ . Therefore, the current notion of the ECBs and UCBs of a task may remain unchanged if a bound on CRPD due to preemption from  $\tau_h$  on  $\tau_i$  is defined as:  $UCB_i \cap' ECB_h$  where the result is a multiset that contains each element from  $UCB_i$  if it is also in  $ECB_h$ , e.g.  $UCB_2 \cap' ECB_1 = \{1, 2, 2, 2\} \cap' \{1, 2\} = \{1, 2, 2, 2\}$ . In case of FIFO and PLRU cache replacement policies, the concepts of useful and evicting cache blocks cannot be applied, as shown by Burguiere et al. [10].

## 6 Evaluation

In this section, we show the evaluation results. The goal of the evaluation was to investigate to what extent the proposed method is able to identify schedulable tasksets upon the analysis of the cache-related preemption delays. We compared the state-of-the-art analyses for CRPD: (*ECB-Union Multiset*), (*UCB-Union Multiset*) methods, and (*Combined Multiset*), with two versions of the proposed method, i.e. (*Partitioning-ver1*) which computes CRPD according to Section 5.3, and the version (*Partitioning-ver2*) which computes CRPD from the worst-case preemption combination, presented in Section 5.7.



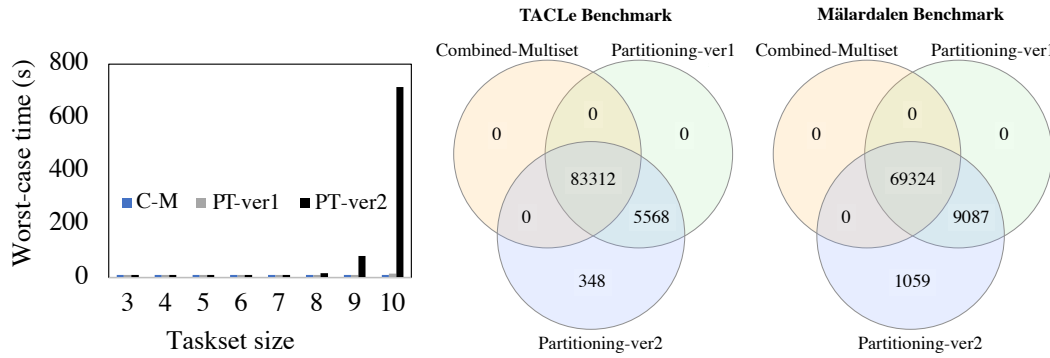


■ **Figure 4** *Left*: Schedulability ratio at different taskset utilisation. *Right*: Weighted measure at different taskset size.

To increase the exhaustiveness of the performed evaluation and the respective results, for the rightmost plots from Figure 4 we used the weighted schedulability measure in order to show a 2-dimensional plot which would otherwise be a 3-dimensional plot, as proposed by Bastoni et al. [7]. In those figures, we show the weighted schedulability measure  $W_y(|\Gamma|)$ , for schedulability test  $y$  as a function of taskset size  $|\Gamma|$ . For each taskset size (in range from 3 to 10), this measure combines data for all of the tasksets generated for each utilisation level from 0.85 to 1, with step of 0.1, since for utilisation levels from 0 to 0.85 all of the compared methods deem almost all tasksets to be schedulable. For each taskset size  $|\Gamma|$ , the schedulability measure  $W_y(|\Gamma|)$  is equal to  $W_y(|\Gamma|) = \sum_{\Gamma} (U_{\Gamma} \times B_y(\Gamma, |\Gamma|)) / \sum_{\Gamma} U_{\Gamma}$ , where  $B_y(\Gamma, |\Gamma|)$  is the binary result (1 if schedulable, 0 otherwise) of a schedulability test  $y$  for a taskset  $\Gamma$  and taskset size  $|\Gamma|$ . Weighting the schedulability results by taskset utilisation means that the method which succeeds to produce a higher weighted measure, compared to the others, is more prone to identify tasksets with higher utilisation as schedulable.

The results show that *Partitioning-ver2* is able to identify more schedulable tasksets compared to the others for any given taskset size, immediately followed by *Partitioning-ver1*. Also, as the taskset size increases, the multiset-based methods deteriorate more in identifying schedulable tasksets compared to the proposed methods. This means that partitioning-based methods are able to identify more tasksets as schedulable with an increase of the taskset size and utilisation.

Next, we report the worst-case computation time results since *Partitioning-ver2* uses set partitioning which is known to be a computation with quadratic/exponential complexity, depending on the algorithm type. The results, reported in the left-most plot in Figure 5,



**Figure 5** *Leftmost:* The worst-case measured analysis time per taskset, at different taskset size. *Center and rightmost:* Venn Diagrams[19] representing schedulability result relations between different methods, over 120000 analysed tasksets per each – TACLe Benchmark and Mälardalen Benchmark.

were computed on MacBook Pro (Retina, 13-inch, Early 2015) version, with Intel Core i5 processor of 2,9 GHz, and DDR3 RAM memory of 8 GB, and 1867 MHz. We used a sequential set partitioning algorithm, and as shown in the graph, in this case exponential complexity is evident for *Partitioning-ver2*. However, the proposed method is intended to be used offline, and its performance can be improved using the algorithm from [14], and even more with parallel computing, e.g. set partitioning algorithms proposed by Djokic et al. [13]. In contrast, *Partitioning-ver1* has a low worst-case time measured for each experiment for different taskset sizes, similar to the the *Combined-multiset* approach.

Finally, we show the relations between the results in Figure 5 (central and rightmost figures). In the central figure, it is evident that all tasksets from TACLe benchmark, that are identified as schedulable by *Combined-multiset*, are also identified as schedulable by *Partitioning-ver1* and *Partitioning-ver2*. However, partitioning-based approaches identify 5568 (and 9087) additional schedulable tasksets depending on the benchmark, while *Partitioning-ver2* identifies additional 348 (and 1059) schedulable tasksets compared to *Partitioning-ver1*. In conclusion of the evaluation, we notice that the proposed partitioning-based algorithms outperform existing state of the art *Combined-Multiset* approach. Also *Partitioning-ver2* outperforms *Partitioning-ver1*, however this comes with the expense of time complexity. The complexity of *Partitioning-ver2* can be further decreased by narrowing down the task interactions for which the preemption combinations should be generated. This remains as a part of the future work as well as the formal proof of the dominance relations between the methods. Finally, the proposed approaches allow for a hybrid, joint use of the two proposed algorithms, while *Partitioning-ver1* significantly outperforms the existing multiset approaches without the expense of time complexity.

## 7 Conclusions

In this paper, we proposed a partitioning based cache-aware schedulability analysis for precise and safe estimation of cache-related preemption delays in the context of fully-preemptive scheduling of real-time systems with sporadic tasks with fixed priorities. The proposed methods are based on a precise analysis of: 1) different preemption subgroups, and 2) different preemption combinations that may occur within a system, and therefore they are able to compute more precise cache-related preemption delay estimations compared to the state of the art approaches. The evaluation was performed using the realistic task parameters

from well-established benchmarks, obtained with a low-level analysis tool, and it showed that the proposed approaches manage to identify significantly more schedulable tasksets compared to the other preemption-cost aware approaches.

In future work, we will apply the proposed method in the context of limited preemptive scheduling since for such task model partitioning-based consideration of preemptions can lead to a more precise computation of cache-related preemption delay. We will also apply the proposed methods to other cache architectures and replacement protocols since many existing analyses inherit the overly pessimistic estimations which are identified in this paper. Finally, we will define a more precise static analysis on number of cache block reloads that are possible during the execution of a task since the existing useful cache block concept significantly over-approximates cache-block reloadability.

---

## References

---

- 1 Martin Aigner. A characterization of the bell numbers. *Discrete mathematics*, 205(1-3):207–210, 1999.
- 2 Sebastian Altmeyer and Claire Burguiere. A new notion of useful cache block to improve the bounds of cache-related preemption delay. In *Real-Time Systems, 2009. ECRTS'09. 21st Euromicro Conference on*, pages 109–118. IEEE, 2009.
- 3 Sebastian Altmeyer, Robert I Davis, and Claire Maiza. Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. In *2011 IEEE 32nd Real-Time Systems Symposium*, pages 261–271. IEEE, 2011.
- 4 Sebastian Altmeyer, Robert I Davis, and Claire Maiza. Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems*, 48(5):499–526, 2012.
- 5 Sebastian Altmeyer, Roeland Douma, Will Lunniss, and Robert I Davis. On the effectiveness of cache partitioning in hard real-time systems. *Real-Time Systems*, 52(5):598–643, 2016.
- 6 Sebastian Altmeyer, Claire Maiza, and Jan Reineke. Resilience analysis: tightening the CRPD bound for set-associative caches. In *ACM Sigplan Notices*, volume 45, pages 153–162. ACM, 2010.
- 7 Andrea Bastoni, Björn Brandenburg, and James Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. *Proceedings of OSPERT*, pages 33–44, 2010.
- 8 Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.
- 9 Tobias Bläß, Sebastian Hahn, and Jan Reineke. Write-back caches in wcet analysis. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- 10 Claire Burguière, Jan Reineke, and Sebastian Altmeyer. Cache-related preemption delay computation for set-associative caches-pitfalls and solutions. In *9th International Workshop on Worst-Case Execution Time Analysis (WCET'09)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.
- 11 José V Busquets-Mataix, Juan José Serrano, Rafael Ors, Pedro Gil, and Andy Wellings. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *Real-Time Technology and Applications Symposium, 1996. Proceedings., 1996 IEEE*, pages 204–212. IEEE, 1996.
- 12 Robert I Davis, Sebastian Altmeyer, and Jan Reineke. Response-time analysis for fixed-priority systems with a write-back cache. *Real-Time Systems*, 54(4):912–963, 2018.
- 13 Borivoje Djokić, Masahiro Miyakawa, Satoshi Sekiguchi, Ichiro Semba, and Ivan Stojmenović. Parallel algorithms for generating subsets and set partitions. In *International Symposium on Algorithms*, pages 76–85. Springer, 1990.
- 14 MC Er. A fast algorithm for generating set partitions. *The Computer Journal*, 31(3):283–284, 1988.



- 15 Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener. Taclebench: A benchmark collection to support worst-case execution time research. In *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2016.
- 16 Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. The malmödalén wcet benchmarks: Past, present and future. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010 .
- 17 Sebastian Hahn, Michael Jacobs, and Jan Reineke. Enabling compositionality for multicore timing analysis. In *Proceedings of the 24th international conference on real-time networks and systems*, pages 299–308. ACM, 2016.
- 18 Paul R Halmos. *Naive set theory*. Courier Dover Publications, 2017.
- 19 Henry Heberle, Gabriela Vaz Meirelles, Felipe R da Silva, Guilherme P Telles, and Rosane Minghim. Interactiven: a web-based tool for the analysis of sets through venn diagrams. *BMC bioinformatics*, 16(1):169, 2015.
- 20 Chang-Gun Lee, Joosun Han, Yang-Min Seo, Sang Luyi Min, Rhan Ha, Seongsoo Hong, Chang Yun Park, Minsuk Lee, and Chong Sam Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Transactions on Computers*, 47(6):700–713, 1998.
- 21 Rodolfo Pellizzoni, Bach D Bui, Marco Caccamo, and Lui Sha. Coscheduling of CPU and I/O transactions in COTS-based embedded systems. In *Real-Time Systems Symposium, 2008*, pages 221–231. IEEE, 2008.
- 22 Harini Ramaprasad and Frank Mueller. Bounding worst-case response time for tasks with non-preemptive regions. In *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 58–67. IEEE, 2008.
- 23 Harini Ramaprasad and Frank Mueller. Tightening the bounds on feasible preemptions. *ACM Transactions on Embedded Computing Systems (TECS)*, 10(2):27, 2010.
- 24 Syed Aftab Rashid, Geoffrey Nelissen, Sebastian Altmeyer, Robert I Davis, and Eduardo Tovar. Integrated analysis of cache related preemption delays and cache persistence reload overheads. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 188–198. IEEE, 2017.
- 25 Syed Aftab Rashid, Geoffrey Nelissen, Damien Hardy, Benny Akesson, Isabelle Puaut, and Eduardo Tovar. Cache-persistence-aware response-time analysis for fixed-priority preemptive systems. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 262–272. IEEE, 2016.
- 26 Altmeyer Sebastian, Douma Roeland, Luniss Will, and I Davis Robert. Evaluation of cache partitioning for hard real-time systems. In *proceedings Euromicro Conference on Real-Time Systems (ECRTS)*, pages 15–26, 2014.
- 27 Darshit Shah, Sebastian Hahn, and Jan Reineke. Experimental evaluation of cache-related preemption delay aware timing analysis. In *18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- 28 Jan Staschulat, Simon Schliecker, and Rolf Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *Real-Time Systems, 2005. (ECRTS 2005). Proceedings. 17th Euromicro Conference on*, pages 41–48. IEEE, 2005.
- 29 Gregory Stock, Sebastian Hahn, and Jan Reineke. Cache persistence analysis: Finally exact. In *Real-Time Systems Symposium (RTSS)*, December 2019.
- 30 Yudong Tan and Vincent Mooney. Timing analysis for preemptive multitasking real-time systems with caches. *ACM Transactions on Embedded Computing Systems (TECS)*, 6(1):7, 2007.
- 31 Hiroyuki Tomiyama and Nikil D Dutt. Program path analysis to bound cache-related preemption delay in preemptive real-time systems. In *Proceedings of the eighth international workshop on Hardware/software codesign*, pages 67–71. ACM, 2000.